CAR-TR-71
CS-TR-1417

DAAK70-83-K-0018
July 1984

AN IMAGE FLOW SIMULATOR

Sarvajit S. Sinha and Allen M. Waxman
Center for Automation Research
University of Maryland
College Park, MD 20742

COMPUTER VISION LABORATORY

CENTER FOR AUTOMATION RESEARCH

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

CAR-TR-71
CS-TR-1417

DAAK70-83-K-0018
July 1984

AN IMAGE FLOW SIMULATOR

Sarvajit S. Sinha and Allen M. Waxman
Center for Automation Research
University of Maryland
College Park, MD 20742

ABSTRACT

The analysis of time-varying images is curently of great interest in computer vision. There has been a deal of work recently in the study of the 2-D image flow produced due to space motion, and the recovery of the object's 3-D structure and space motion from the flow. This report details the reverse process implemented in the form of an Image Flow Simulator; from a knowledge of stucture and motion, to display the 2-D image sequence and associated flow. This 3-D graphics animation package simulates motion of objects through space and also the evolution of surface contours through time. The graphics algorithms for projection, clipping, hidden surface removal, shading and animation are described in this report.

The support of the Defence Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under Contract DAAK70-83-K-0018 (DARPA Order 3206) is gratefully acknowledged.

1. INTRODUCTION

This technical report describes the Image Flow Simulator (IFS version 1) developed at the Computer Vision Laboratory of the Center for Automation Research. As the name suggests, the package is a software tool for research and experimentation in the areas of time-varying imagery, image flow and motion, specifically in the context of the overall "Image Flow Paradigm" as described by Waxman[7]. This report will not attempt to address the motivations for this approach to motion analysis, which have been dealt with in [7], but will describe the graphics algorithms required in building the simulator and examples of its use.

The simulator consists of a menu-driven, three dimensional graphics subsystem with certain structural primitives built in. It allows the user to create a scene in 3-D space, define initial positions of each object in space, define observer motion parameters and view the time-varying imagery as the observer moves through space. The simulator also allows the user to define feature points and edge-contours on objects in the scene and track them over time. At each time instant, the user may visualize the full image flow produced over the whole image plane of the observer, or at specific points in the image plane. At the end of the simulation, the user may also draw the image space-time stream tube associated with each image contour[7]. The simulator also outputs the values of the image flow velocities, and image plane coordinates of feature points at each time instant; information that can later be utilized for other experiments such as those

those in Dynamic Stereo[8].

The key ideas in construction of this simulator are simplicity, robustness and modularity. Care has been taken to maintain simplicity of algorithms in order to keep the simulation time to a minimum, since it is interactive. Robustness is important in that the user has been allowed a great deal of freedom in setting up a scene, with occlusion and insertion (one primitive partially within another) possible. Being menu driven in its present form, the simulator is somewhat limited, but its modular nature allows new graphical primitives and algorithms to be inserted easily. For example, a future extension should allow for independent rigid body motions to be assigned to each object in the scene, as well as to the moving observer.

Section 2 describes the scene and object primitives, and the coordinate systems used in the system. Section 3 discusses the various geometric transformations needed for object display in static scenes and for time-varying images. The graphics routines for hidden-surface elimination and shading are discussed in Section 4. Section 5 describes the image flow and stream tube display subsystems. A few sample runs are included in the Appendix in order to show the capabilities of the Image Flow Simulator, as well as serving as an instructional guide to future users. The source code listings, written in "C", are archived on CVL tapes along with this report.

2. COORDINATE SYSTEMS AND SOLID MODELLING

The observer's reference frame is represented by the right handed coordinate system shown in Figure 1. The observer is located at the origin, such that the center of field of view is along the Z-axis. The two dimensional image is formed by perspective projection of points in the scene onto a planar screen oriented normal to the viewing direction, such that the image (x,y) origin is located at $(X,Y,Z)=(0,0,f)$. This is similar to an image formed in a pin-hole camera of focal length f being reinverted and reversed. The image plane size is variable, extending $\frac{-imagesize}{2} \leq (x, y) \leq \frac{imagesize}{2}$. Since the image coordinates are scaled by the focal length in the process of perspective projection, we shall now refer to x, y as the scaled directions rather than image distances. Thus there exists a "viewing cone"; images of objects lying outside this region are clipped.

There are a number of coordinate systems used to refer to the image. The system magnifies the image produced above into an internal picture buffer (whose size is specified by the user) when drawing scenes or flow vectors, and when complete, puts it in the display device window using device (Grinnell) specific routines. The words "picture buffer" and "work window" or "device window" refer to the same scene, one to the internal workspace, the other to the window reserved on the Grinnell for picture output. Thus, the image is referenced in a variety of ways.

1. Observer's image coordinates (the observer screen size being set at the beginning of the simulation process),

2. Picture-buffer coordinates (same as 1. except the image coordinates are magnified to the output picture size and the origin is at the lower right of the picture). This is related to the observer coordinates by

$$x_{picture\text{-}buffer} = magnification\text{-}factor \times x + \frac{picture\text{-}size}{2},$$

$$y_{picture\text{-}buffer} = magnification\text{-}factor \times y + \frac{picture\text{-}size}{2},$$

$$where \;\; magnification\text{-}factor = \frac{picture\text{-}size}{imagesize}.$$

All calculations of shading, hidden surface removal, and flow vector drawing are done in this coordinate system.

3. Picture-buffer array reference values (origin is at left top of picture). This is related to 2. by

$$x_{array} = picture\text{-}size - x_{picture\text{-}buffer} - 1,$$

$$y_{array} = picture\text{-}size - y_{picture\text{-}buffer} - 1.$$

4. Absolute device (Grinnell) coordinates (origin is at left bottom of picture). This is related to 2. by

$$x_{grinnell} = picture\text{-}size - x_{picture\text{-}buffer} - 1,$$

$$y_{grinnell} = y_{picture\text{-}buffer}$$

The instantaneous rigid body motion of the observer's coordinate system is specified in terms of the translation velocity $\mathbf{V} = (V_X, V_Y, V_Z)$ of its origin and its rotational velocity $\mathbf{\Omega} = (\Omega_X, \Omega_Y, \Omega_Z)$.

2.1 SOLID OBJECT MODELS

Each object primitive in the system has a lattice of control points that describes its shape and is defined in its own coordinate system. Figure 2 shows these primitive structure descriptions for each of the shapes currently available in the system. For each primitive selected, the user must provide characteristic measurements : the RADIUS of the sphere, the RADIUS of the base and the HEIGHT of the cone, the LENGTH and RADIUS of the cylinder, the LENGTH, BREADTH, and HEIGHT of the cuboid, the exents of the quadratic surface in the X-Y plane.

The sphere is constructed by sweeping the angles θ (longitude) through 0 to 360 degrees and ϕ (latitude) through -90 to +90 degrees at a constant interval. The poles are located on the Y-axis at $\pm$ RADIUS, and $\theta = 0$ corresponds to the X-axis. This produces an ordering of the polygons forming the surface such that (at least for small angles of rotation of the primitive) the visible polygons will be drawn first, saving computation when deciding about hidden surfaces.

The cone is modelled as triangles, the tip of the cone being fixed at its HEIGHT, the base being on the X-Z plane. The base is swept at regular intervals of θ, producing triangles for the shading algorithm.

The cylinder is formed by fixing the height at $\pm$ LENGTH/2, and producing strips along the sides of the cylinder by varying θ from 0 to 360 degrees. Each strip is subdivided into two triangles and shaded. The top and bottom are closed by creating triangles from the centers at the top and bottom and the control

points along the sides.

The cuboid's sides and the quadric surface (which includes a plane) are modelled similarly, scanning fixed intervals in two sides to produce a grid. Here again, four neighbouring control points are subdivided into triangles and sent to the shading routine.

It must be noted that although the resultant surface is only a polygonal approximation, the control points themselves are exact points. Also, the subdivision into triangles in the manner described above does not give rise to inconsistencies such as surface patches not being continuous at edges.

Surfaces are approximated by polyhedra for reasons of speed. A best fit quadric surface or spline interpolation was not used for describing the surface since these involve much more computation. The polyhedral approximation leads to some errors in the simulation, but this is minimized by other methods later in the process.

As can be seen, it would not be difficult to introduce other primitive shapes or modify existing ones (e.g. change the sphere to a general ellipsoid). There is a local coordinate frame associated with each primitive, the one it is defined in, and this is implicitly maintained throughout the simulation process. Operations such as rigid body motions on an object are actually performed by applying appropriate transformation matrices on the control points. It is assumed that each object is a rigid body in order to preserve the same structural relationship between the control points as was originally defined, i.e., the control points do not move in

their local coordinate system.

Initially, each object's local coordinate system is coincident with the observer's system. The user is then allowed to place the objects out in space and thus build up a scene. The operator is given the option of seeing the scene from the observer's position, or from some other point in space. For the second option, he must provide the new viewing position and direction. The coordinate transformation matrices for these changes are described below. Similarly, the changes in the scene due to relative viewer motion is described in terms of a transformation matrix later.

It is helpful to think of the scene as a tree structure, the root being the whole scene itself, and the branches being the types of primitive objects available to the user. Each leaf on the tree stores the characteristic measurements of each primitive. The system stores information regarding the exact position of each object in the observer's coordinate system by also keeping the latest description of the control point structure. Figure 3 shows this hierarchical development of the scene. Complex objects, such as that shown in Figure 4, can be built up from the structural primitives, although it does require a bit of preplanning to set each sub-part in the correct position, as there are so many parameters to be set (size, orientation and position). It is important to note that our process of "scene construction" does have an underlying notion of "attachment" of one object to another. This is implied by the sub-scene relationships. Unlike Sugihara's approach to animation[5], each object is not treated with independent transformation matrices. New scenes, from different viewing positions over time, are

obtained by applying the same transformation matrix over every object in the scene. Therefore, relative motion between objects is not possible with the current version of the simulator. It is one of the primary extensions to be incorporated into the next version of the system.

3. GEOMETRIC TRANSFORMATIONS

After an object is defined in its own coordinate system, it is "placed" into the world coordinate frame by multiplying each control point by a transformation matrix. This placement is specified by entering the rotations of the object around each world axis (while it is located at the world origin), followed by its displacement into the world. Obviously the displacement in the Z-direction must be greater than f, since that is the focal length of the system. Internally, a rotation of the object by θ around any axis is equivalent to a rotation of the world frame by $-\theta$. A translation of the object frame by (T_X, T_Y, T_Z) is equivalent to the world frame moving $(-T_X, -T_Y, -T_Z)$.

Each object point (X_o, Y_o, Z_o) is represented in homogenous coordinates as $(X_o, Y_o, Z_o, 1)$. The transformation matrix is a 4×4 matrix and is a product of standard transformation matrices R_X, R_Y, R_Z about the X_w, Y_w, Z_w axes summed with a translation vector (T_X, T_Y, T_Z) (these can be found in [1] or [3]). The matrix product is created internally from user supplied $\theta_X, \theta_Y, \theta_Z, T_X, T_Y, T_Z$. The sequence of axes around which rotations occur is fixed as given below. We have

$$(X_w, Y_w, Z_w) = (X_o, Y_o, Z_o) \cdot R_X(-\theta_X) \cdot R_Y(-\theta_Y) \cdot R_Z(-\theta_Z) + (-T_X, -T_Y, -T_Z).$$

Once the object has been placed in the world, it can be viewed from any point in space. If the camera is located at the origin of the world as shown in Figure 1. then the image coordinates are immediately available as a perspective

transform :

$$\frac{x}{f} = \frac{X_w}{Z_w} \quad , \quad \frac{y}{f} = \frac{Y_w}{Z_w}.$$

However, if the operator wishes to view the scene from some other point in the world, each control point must again be transformed into the new observer coordinate frame. The user specifies the new eye coordinates in the world $(X_e\ ,\ Y_e\ ,\ Z_e)$ and the position of a point he is looking at $(X_a\ ,\ Y_a\ ,\ Z_a)$. The transformation is set up as shown:

$$(X_w' \ ,\ Y_w' \ ,\ Z_w' \ ) = (X_w - X_e\ ,\ Y_w - Y_e\ ,\ Z_w - Z_e\ ) \cdot R_X(\theta_1) \cdot R_Y(\theta_2)$$

$$\text{where } \theta_1 = \arctan\left[\frac{Y_a - Y_e}{\sqrt{(Z_a - Z_e)^2 + (X_a - X_e)^2}}\right]$$

$$\text{and } \theta_2 = \arctan\left[\frac{X_a - X_e}{Z_a - Z_e}\right]$$

Then a perspective projection can be taken in the new (primed) coordinate system. Note that one degree of freedom remains, i.e. rotation around the new Z axis is unfixed and therefore the images produced may be rotated at an arbitrary angle in this case.

After projection, the image coordinates are magnified into the picture or device-window coordinates.

3.1 RIGID BODY MOTION

IFS creates time-varying images by assigning a velocity to the observer and tracking each control point in the scene as it moves relative to the camera. Motions of objects in the scene are simulated by assigning the relative motion to the camera. Then, assuming the time interval between the frames is sufficiently short, the position of a control point in each frame can be described in terms of the translational velocity V_X, V_Y, V_Z and rotational velocity Ω_X, Ω_Y, Ω_Z. Both terms are assumed to be small, so that one can approximate the derivative as a difference. Thus the displacement due to the translational velocity is given by

$$\vec{X}' = \vec{X} + \vec{V}\,\delta t\,.$$

For infinitesimal rotations, we can represent the rotation angles as vectors and approximate

$$\delta\vec{\omega} = \vec{\Omega}\,\delta t\,,$$

where $\delta\vec{\omega} = \delta\omega_X\,\hat{i} + \delta\omega_Y\,\hat{j} + \delta\omega_Z\,\hat{k}\,.$

Infinitesimal rotations are commutative, and therefore interchangeable. The apparent motion of a point due to an infinitesimal rotation of the camera is given by [2]

$$\vec{X}' = \underline{R}\,.\,\vec{X} \qquad \text{where} \qquad \underline{R} = \underline{I} + \underline{\epsilon}$$

$$\text{and } \underline{\epsilon} = \begin{bmatrix} 0 & \delta\omega_Z & -\delta\omega_Y \\ -\delta\omega_Z & 0 & \delta\omega_X \\ \delta\omega_Y & -\delta\omega_X & 0 \end{bmatrix}.$$

The composite matrix of both translation and rotation in homogenous coordinates is given by

$$(X' \ , \ Y' \ , \ Z' \ \ 1) = (X, Y, Z) \cdot \begin{pmatrix} 1 & \delta\omega_Z & -\delta\omega_Y & 0 \\ -\delta\omega_Z & 1 & \delta\omega_X & 0 \\ \delta\omega_Y & -\delta\omega_X & 1 & 0 \\ V_X & V_Y & V_Z & 1 \end{pmatrix} \delta t$$

During the simulation of motion over time, the observer's rigid body motion parameters are held constant in his own evolving reference frame. A possible extension for the future would be to incorporate simple dynamical models governing the motion of objects in the scene as well as motion of the observer.

4. GRAPHICS ROUTINES

There are two major components of the graphics routines: removal of hidden surfaces and quasi-realistic shading. The input to these two routines is a polygon formed by three points in the world coordinate system as described in Section 2.1. The problem of removal of occluded surfaces has been solved in many elegant algorithms, most involving some kind of sorting of the polygons forming an object [6]. These algorithms were not used since time varying images change the ordering of the polygons. This would require the data structure to be sorted at each frame, a time-consuming process. The following table from [1] shows the relative effectiveness of four hidden-surface removal algorithms.

Algorithm	Number of Polygonal faces		
	100	2500	60000
Depth sort	1*	10	507
Z-Buffer (IFS)	54	54	54
Scanline	5	21	100
Warnock Area Subdivision	11	64	307

*Values scaled so that this is 1.

The most effective algorithm for our purposes was found to be the "Z-Buffer" method, since a typical scene in IFS has about 3000 polygons. This method is an extremely simple one, that of keeping a two-dimensional array whose content is the Z-distance associated with each pixel of the picture. A point in space is visible if, after perspective transformation, the Z-distance associated with the image pixel is less than the contents of the corresponding location in the Z-buffer. This requires keeping a floating point array the size of the output picture; for a 256×256 picture memory on the order of 1MB is required. However, it turns out

to be useful to keep a Z-depth map for the later calculations involving image flow. Also, depth-sort methods generally do not allow for penetrating polygons, a necessary feature in IFS. Algorithms utilizing frame-to-frame coherence of images such as [4] are available, but are not robust enough for our requirements, allowing only camera motion, and thus incompatible with planned extensions of the simulator.

Shading and depth mapping are done simultaneously in picture coordinates (in scan-line order) by first interpolating between the control point image pixels to the current scan line, and then interpolating between the two boundary values (determined by the triangle and the scan line) to set the shade and Z value for individual pixels. The regions of the image outside the picture frame are "clipped" against the edges of the device window at the same time that the polygons are shaded.

Each control point of a polygon in world-coordinates is transformed into image coordinates (floating point) and then expanded and rounded into the device frame pixels (integer). Initially, the Z-distance associated with this pixel in the Z-Buffer was taken to be that of the associated control point; however, this leads to errors due to roundoff. In order to find the correct value of Z corresponding to this pixel, a ray emanating from the origin of the world coordinate system and passing through the pixel's image coordinate is backprojected onto the plane formed by its three associated control points. The Z value is found from the point of intersection of the ray and the plane, the equation being solved by Cramer's rule. The solution is unstable for a plane that is nearly tangential to the ray; this

condition is dealt with as a special case. The backprojection procedure is repeated for each of the three control points, and the Z-map built up by linear interpolation when the scene is shaded.

Shading models attempt to recreate the effects of lighting in the scene. The simplest is to model the surfaces as exhibiting diffuse reflection (such that it appears to have the same brightness from any viewing position) and the light source as a plane with rays parallel to the Z-axis. The intensity of light calculated by Lambert's cosine law is

$I = k\,\vec{L}\cdot\vec{N}$ where $\vec{L}$ is the direction of the light source, k is the diffuse reflection coefficient, and $\vec{N}$ is the surface normal.

IFS models the light source as a point source and allows the operator to position it anywhere in the world. A point source of light yields spherical light waves impinging on the scene; however, we do not take into account the inverse-square degradation of intensity with distance. Such lighting allows for more realistic shading of planar surfaces in space. For each polygon, we find the surface normal by taking the cross product of the vectors along two sides, and calculate the intensity as the inner product as shown above. The coefficient k is chosen to be 1, and the value scaled such that I=1 corresponds to white (gray level 255) and I=0 to black (gray level 0). In this manner we find the intensity of each polygon. Note that for diffuse reflection, the intensity depends only upon the angle between the surface normal and the light source, not on viewing direction. The simulator does not at present allow for specular reflection or shadowing

effects. More complex algorithms for shading can be found in [1].

The fact that one finds the angle between the surface normal and the light source is also used to speed up the process of hidden surface elimination. If the light source is placed at the origin, then for closed convex polyhedra (all the primitive objects except the quadric surface) a polygon is visible only if this angle is between $\pm 90°$. If the angle is greater than these values, then the surface is obviously facing away from the light source, and hence the camera origin, and therefore is not processed further. This method can only be used if the source is positioned at the origin.

5. FEATURE POINTS, CONTOURS AND IMAGE FLOW

Feature points and contours can be defined on the objects in the scene and displayed as the scene changes over time. They have been included in order to allow visualization of image contour deformation and neighborhood evolution (Figs. 5,6). A contour is approximated as straight line segments between vertices, stored in an appropriate data structure. Therefore, both individual feature points and curve vertices can be stored in the same structure. Currently, IFS handles three ways of defining the feature points and contours:

1. Interactively using the trackball and Grinnell-specific routines. The cursor is located at the desired position of the feature (or vertex), and the point is entered. IFS finds the relative location of the point in the device window coordinates, and converts from this to the image plane and to the world coordinate system by once again finding the range in the Z-buffer and reversing the perspective transform. The user is given the safeguard of trashing the value in order to recover from mistakes.

2. From a file (in predetermined format) containing the image coordinates and other data regarding the feature (being individual feature points or a curve) and the number of points (vertices for contours) in the feature. The actual world coordinates corresponding to these image coordinates is calculated by extracting the Z-distance from the Z-buffer, and then reversing the perspective transform. The format for this file is given below:

```
<0/1>     <n>      /*Feature type and number of feature points/vertices
<x1>      <y1>     /*0=Individual feature points, 1=feature curve
<x2>      <y2>     /*
:    :             /*x,y coordinates of features in image coordinates
:    :
<xn>      <yn>
<0/1>     <n1>          /*repeat for next feature
:    :
:    :
```

3. From a file containing the world coordinates of the features (this file is written in binary) which is created by IFS on demand for experiments such as those desired for studying base-line effects on *Dynamic Stereo.*

The world coordinates of a feature point are used in order to find the location of the feature point at different time steps. The transformation matrices used are the ones described in Section 3. There is one difference between the treatment of a feature and that of an object control point; the feature point is "painted" on the surface in the scene by projecting the image coordinates of the features into the world and then joining them by straight line segments. These segments are not compared against the Z-buffer to find occlusion, since joining feature points across polygon boundaries would cause the segments to be occluded. However, the present method also has drawbacks in cases where a genuine occlusion (due to observer motion, perhaps) is not displayed. This problem could be rectified by setting a distance threshold for occlusion of contours.

When defining the contours and feature points interactively through the trackball, the trackball must be set to "cursor#1" (the program is only set up to read this cursor) and must be set in "point" mode(not "track"). The color of the cursor (black or white) can be changed from the trackball depending upon the

background.

The current version of the simulator only allows for features and contours to be placed on the initially visible portions of the object surfaces. Future extension will allow for all sides of objects to be "decorated" before they are moved out into the scene.

5.1 IMAGE FLOW

As a rigid object or the monocular observer moves through space, an image sequence or flow field is created on the observer's focal plane. This flow is determined by the parameters of space motion and the structure of objects in the scene. The flow is mathematically defined as the time derivative of the image coordinates of a feature due to the motion of the corresponding point in 3D space. The flow equations are taken from Waxman and Ullman [9]. At each time step of the simulation process, the user may find the full image flow vectors either at the specified feature points on the surface or on a grid over the whole image. Since both the relative motion parameters of the observer and the parameters of the objects in the scene are known, computation of the flow is straightforward; it is given by the set of equations

$$v_x = \left\{ x \frac{V_Z}{Z} - \frac{V_X}{Z} \right\} + [\, x\, y\, \Omega_X - (\, 1 + x^2\,)\, \Omega_Y + \, y\, \Omega_Z \,]$$

$$v_y = \left\{ y \frac{V_Z}{Z} - \frac{V_Y}{Z} \right\} + [\, (\, 1 + y^2)\, \Omega_X - x\, y\, \Omega_Y - \, x\, \Omega_Z \,]$$

Although the system allows the focal length and image size to be variable, these

values must be set to 1 for image flow calculations. As expected, the flow values obtained through the simulator are not exact due to the interpolated Z-buffer, but have been found to be adequate for experiments carried out in [8]. For a plane, the error has been found to be on the order of 0.1 percent.

The positions where the flow is calculated are found in picture coordinates (integer), so that one can also access the range value using the same coordinates in the Z-buffer. The Z-buffer is sampled at intervals of some user-specified size if the user wishes to display the flow field as a grid over the image (option 1), or is found at the image coordinates corresponding to the features defined earlier (option 2). These integer coordinates are later converted into the x,y coordinates of the image plane. All quantities in the above equations are specified, and the full flow can easily be constructed. Examples are shown in Figure 6.

The flow field is displayed as vectors at the points where it is calculated, the vector length being scaled to either the grid spacing in option 1 or to some user specified length in option 2. Bresham's algorithm is used for line drawing since it is a highly efficient line drawing routine involving fewer floating point operations than the usual y=mx+c approach [1]. However, no attempt is made at antialiasing the line to remove jaggies. Both the drawing of the small curve segments and the flow vectors use the same algorithm. The final clipping of lines against the device window edges occurs as in the shading process.

The flow values and the corresponding image coordinates at which they are calculated are then stored in a specified file for later use. In its current version, the simulator does not put flow values of different features into separate files, and

only finds the full flow (as opposed to the normal flow) at vertices of the feature contour.

5.2 STREAM TUBES

At the end of the simulation process, IFS allows the user to create the stream tubes corresponding to the evolving contours [7]. The stream tube is an image space-time representation of the deformation of a contour in an image (see Fig. 5). It is a three dimensional representation, two axes representing the x,y of the image plane, while the third axis represents time. The stream tube is drawn individually for each of the contours in the scene. The image of the curve in the world coordinates is found by perspective projection and once again placed in front of the eye through a transformation matrix given by

$$\begin{pmatrix} \cos(45^o) & \sin(45^o) & 0 & 0 \\ -\sin(45^o) & \cos(45^o) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \text{stackspace} & 50 & 0 \end{pmatrix}$$

This is done for the image of each of the contours at each time instant and the images separated along the t-axis by the "stackspace". This quantity is a user specified value representing the distance in graphic units between the images at each time step. The curve x,y coordinates in the image at each time instant are automatically stored in a specified file. The option of drawing a stream tube should only be used for feature contours in the image plane, not for feature points. As a last option in the simulator, IFS also allows the user to visualize the motion of each of the vertex points of the contour over time, i.e., each contour

vertex is connected by straight line segments to its image in the next time frame.

It must be noted that the system assumes that the stream tube is being drawn for the feature contours only, not for the feature points. If it is given a set of feature points to draw a stream tube of, it may attempt to join up the points in an arbitrary fashion. However, the final option, to join feature points/contour vertices over time, may be used in either case.

6. PORTABILITY AND EXTENSIONS

The current version of the Image Flow Simulator has been written in "C" for the Computer Vision Laboratory's VAX 11/780 running Unix 4.1BSD. It uses a number of site specific routines, but these are isolated in a few subroutines that are mentioned in the listing of the package. They are related to the available Grinnell graphics system, and to the configuration at the CVL. The package of routines available in the library "-lgr" is used to initialize the work window on the Grinnell; to put the internal work buffer onto the display; to control the cursor and trackball while defining features; and to draw line segments while creating the feature contours. These are the only routines which would need to be replaced at other installations running a UNIX/C environment.

6.1 EXTENSIONS

1. The simulator should be converted into a graphics language, allowing the user to create his own simulation stream, thus removing the drawback of being menu driven. This would require only minor modifications due to the modular nature of the current version. The graphics core is already present in the current version; a language would merely require an interpreter.

2. IFS should allow for independent rigid body motions of each object in the scene, rather than restricting motion to only the camera as is currently the case. This would require a slight modification in the data stucture representing the scene so that a transformation matrix can be associated with each object in the scene. This would make the IFS similar to the graphics animation system of

Sugihara [5].

3. It should be possible to "paint" features on *all* faces of objects at the start of simulation, not merely on the visible areas. The features should have a range explicitly stored with the control points in the data structure, and their occlusion should also be calculated and displayed.

7. CONCLUSIONS

This report has described the first version of the Image Flow Simulator currently in use by the motion research group at the Computer Vision Laboratory. It allows the user to quickly visualize the effects of observer motion with respect to a static scene, and the evolution of feature points and contours in that scene. This should be of help in understanding the concepts of neighborhood deformation, contour evolution, image flow and other ideas associated with time-varying imagery. We expect that the simulator will prove to be a useful tool in motion studies.

We wish to acknowledge the resources provided at the Computer Vision Laboratory of the Center for Automation Research in the development of the simulator.

APPENDIX

The Command sequence for executing the Image Flow Simulator program is:
IFS internalbuffer flowvalues features
The three files required in the command line are:
internalbuffer - a file for working on the picture buffer internally.
flowvalues - contains the image flow values calculated and the image coordinates at each of the points.
features - contains the coordinates of the feature points/vertices of the contours initially. If the feature is not to be defined through a file, then a dummy file name must be used. At the end of a simulation run this file contains the image coordinates of the points at each time instant. The initial contents of this file are destroyed.
A sample run is shown below. User responses are indicated in italics.

****

Image Flow Simulator [version 1]
Do you need help?*yes*

This program allows you to create time varying images
of objects in 3-d space. You must specify the objects
in the scene and their parameters, and the viewer
motion parameters. Perspective projection is used to
create the image, so an object of size 10 units at a
distance of 100 from the observer will produce the same
image as one of size 1 at a distance of 10 units
...More...? *yes*
Imagine that the screen is +/- 0.5 units in x and y,
and at a distance of 1 unit from you in Z (this image is
then expanded to the Grinnell screen size.) Thus the angle
of viewing is about +-30 degrees. The coordinate system
shown above always moves with the observer, so that you will
always be looking down the z-axis. Note that due to relative
motion, if you specify a positive velocity in x, the image
will move to the right(in the negative x direction).

Enter Grinnell window size(integer)200

```
Enter the coordinates of the lower left hand corner
of the Grinnell window to be used
column value = 0
row    value = 0


Enter focal length of camera unit(commonly 1) : 1


Enter image plane size(commonly 1) : 1

Options:
0: Fast algorithm, light source fixed at origin.
1: Light source position variable      .
Choice : 1

Specify position of point light source.
X coordinate of light source =50
Y coordinate of light source =50
Z coordinate of light source =0

Menu :
Choose objects in scene(upto 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number  :1

Cone origin is at center of base
Height of cone = 20
Radius of cone = 10
Euler angle of cone (in integer degrees).
  Rotation about  x(horizontal) axis   = 0
                  y(vertical)   axis   = 0
  Rotation about  z(horizontal) axis   = 0
Where would you like to place the cone?
enter the x-coordinates of the cone origin -5
enter the y-coordinates of the cone origin 25
enter the z-coordinates of the cone origin 110
```

You have the option of seeing the scene from the
observer's point of view or from another point in space.
Will you see observer's view? Your reply must be either yes or no.
yes
cone drawn

see fig 6 a

Delete object from scene?
no
Want to change light source position for next display?*no*

Choose objects in scene(upto 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number :2

Cylinder is drawn from +length/2 to -length/2
Length of cylinder = 30
Radius of cylinder = 10
Euler angle of cylinder (in integer degrees).
Rotation about x(horizontal) axis = 0
y(vertical) axis = 0
Rotation about z(horizontal) axis = 0
Where would you like to place the cylinder?
enter the x-coordinates of the cylinder origin -5
enter the y-coordinates of the cylinder origin 10
enter the z-coordinates of the cylinder origin 110

You have the option of seeing the scene from the
observer's point of view or from another point in space
Will you see observer's view? Your reply must be either yes or no.
yes
cylinder drawn

see fig 6 b

Delete object from scene?
no
Want to change light source position for next display?*no*

Choose objects in scene(upto 5 of any one type)
0 ---> to terminate object creation loop

1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number :3

Parallelepiped is located (0,0,0),(length,0,0)
(0,0,breadth),(0,height,0).....
Length of parallelepiped(in x) =20

Breadth of parallelepiped(in z) =20

Height of parallelepiped(in y) =20
Euler angle of cube (in integer degrees).
Rotation about x(horizontal) axis = -10
y(vertical) axis = 25
Rotation about z(horizontal) axis = 0
Where would you like to place the cube?
enter the x-coordinates of the cube origin -15
enter the y-coordinates of the cube origin -15
enter the z-coordinates of the cube origin 65

You have the option of seeing the scene from the
observer's point of view or from another point in space
Will you see observer's view? Your reply must be either yes or no.
no

Specify new viewing position:
x value in old coordinate system = 50
y value in old coordinate system = 20
z value in old coordinate system = 10

Specify viewing direction by providing coordinates of
some point you are looking at.
x value of this new point in old coordinate system 0
y value of this new point in old coordinate system 0
z value of this new point in old coordinate system75

cone drawn
cylinder drawn
rectangular parallelepiped drawn

_____*see fig 6 c.*

Delete object from scene?
Your reply must be either yes or no.
no
Want to change light source position for next display?*no*

Choose objects in scene(upto 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number :4

Center of sphere is at origin
Radius of sphere =15
Euler angle of sphere (in integer degrees).
Rotation about x(horizontal) axis = 0
y(vertical) axis = 0
Rotation about z(horizontal) axis = 0
Where would you like to place the sphere?
enter the x-coordinates of the sphere origin 5
enter the y-coordinates of the sphere origin -10
enter the z-coordinates of the sphere origin 85

You have the option of seeing the scene from the
observer's point of view or from another point in space
Will you see observer's view? Your reply must be either yes or no.
yes
cone drawn
cylinder drawn
rectangular parallelepiped drawn
sphere drawn

see fig 6 d

Delete object from scene?
no
Want to change light source position for next display?*yes*

Specify position of point light source.
X coordinate of light source =0
Y coordinate of light source =0
Z coordinate of light source =0

Choose objects in scene(upto 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number :5
Bounding limits of surface(max+/-49):
minimum X: -30
maximum X: 30
minimum Y: -30
maximum Y: 30
Enter the coefficients of the 2nd order surface
in the order Z = aX^2+bXY^+cY^2+dX+eY+f
a = 0
b = 0
c = 0
d = 0
e = 2
f = 100
You have the option of seeing the scene from the
observer's point of view or from another point in space
Will you see observer's view? Your reply must be either yes or no.
yes
cone drawn
cylinder drawn
rectangular parallelepiped drawn
sphere drawn
surface drawn

see fig 6 e

Delete object from scene?*no*
Want to change light source position for next display?*no*

Choose objects in scene(upto 5 of any one type)
0 ---> to terminate object creation loop
1 ---> Cone
2 ---> Cylinder
3 ---> Parallelepiped
4 ---> Sphere
5 ---> Surface
6 ---> Help function
Choice of object number :0

Would you like to define feature points on the image?
Your reply must be either yes or no.
yes

Options:
0: You can define features interactively using the trackball
1: You can define features by reading image coordinates from a file
2: You can use the feature points saved from options 1 or 2 in a
previous simulation run
Choice?1
Want to define other curves(using trackball)? Your reply must be either yes or *no*
Do you wish to save the feature point coordinates for later use? *no*

Specify the viewer motion parameters.
Translational velocities in units/time step
Vx = 1
Vy = 2
Vz = 10
Rotational velocities in radians/time step
Ox = 0.08
Oy = 0.09
Oz = 0.1
Specify number of time instants after which view is required : 1

Would you like to see the scene at time 0 on the Grinnell?
Your reply must be either yes or no.
yes
cone drawn
cylinder drawn
rectangular parallelepiped drawn
sphere drawn
surface drawn
feature points drawn
Save the scene?
yes
Enter the filename in which to save : scene.1

see fig 6 f

Would you like to see the flow at time 0 on the Grinnell?
Your reply must be either yes or no.
yes

Options:
0 : You can find the flow at the (individual) feature points.
1 : You can find the flow on a grid over the whole image.
Choice? 0

Interval for plotting flow vectors (in pixels)= 20
Save the scene?
Your reply must be either yes or no.
yes
Enter the filename in which to save : *flow*.1

_____*see fig 6 g*

cone drawn
cylinder drawn
rectangular parallelepiped drawn
sphere drawn
surface drawn
feature points drawn
Save final scene?
Your reply must be either yes or no.
yes
Enter the filename in which to save : *scene*.2

_____*see fig 6 h*

Would you like to see the stream tube?
Your reply must be either yes or no.
no

****

The user specifies the maximum number of time instants of the simulation run (in the above example, this has been set to 1) and the simulator executes from $t = 0$ till this time step. The final scene is always drawn (see Fig. 6h). The simulator then draws the stream tube if necessary and also displays the motion of each individual point through image space-time.

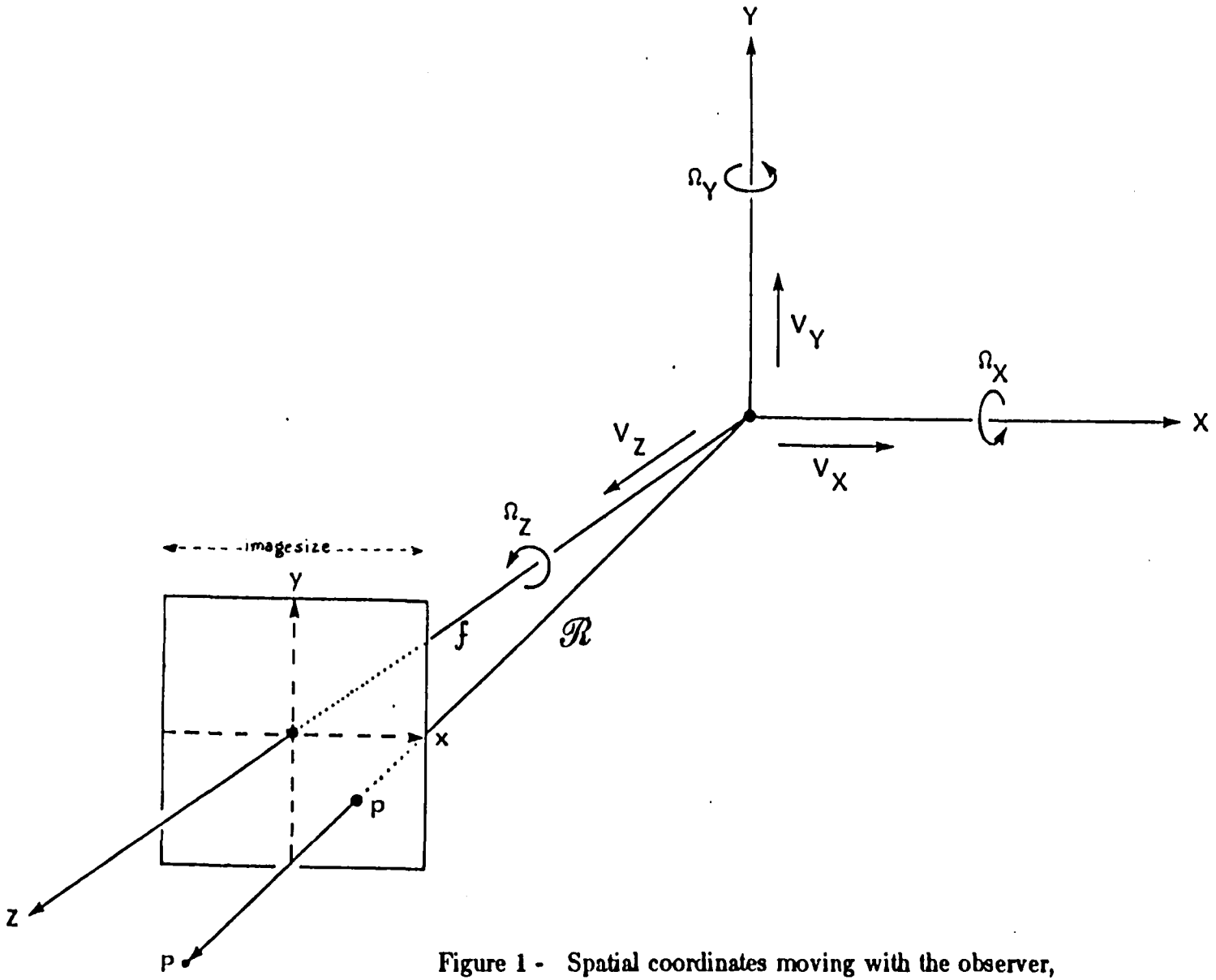

Figure 1 - Spatial coordinates moving with the observer, and image coordinate system.

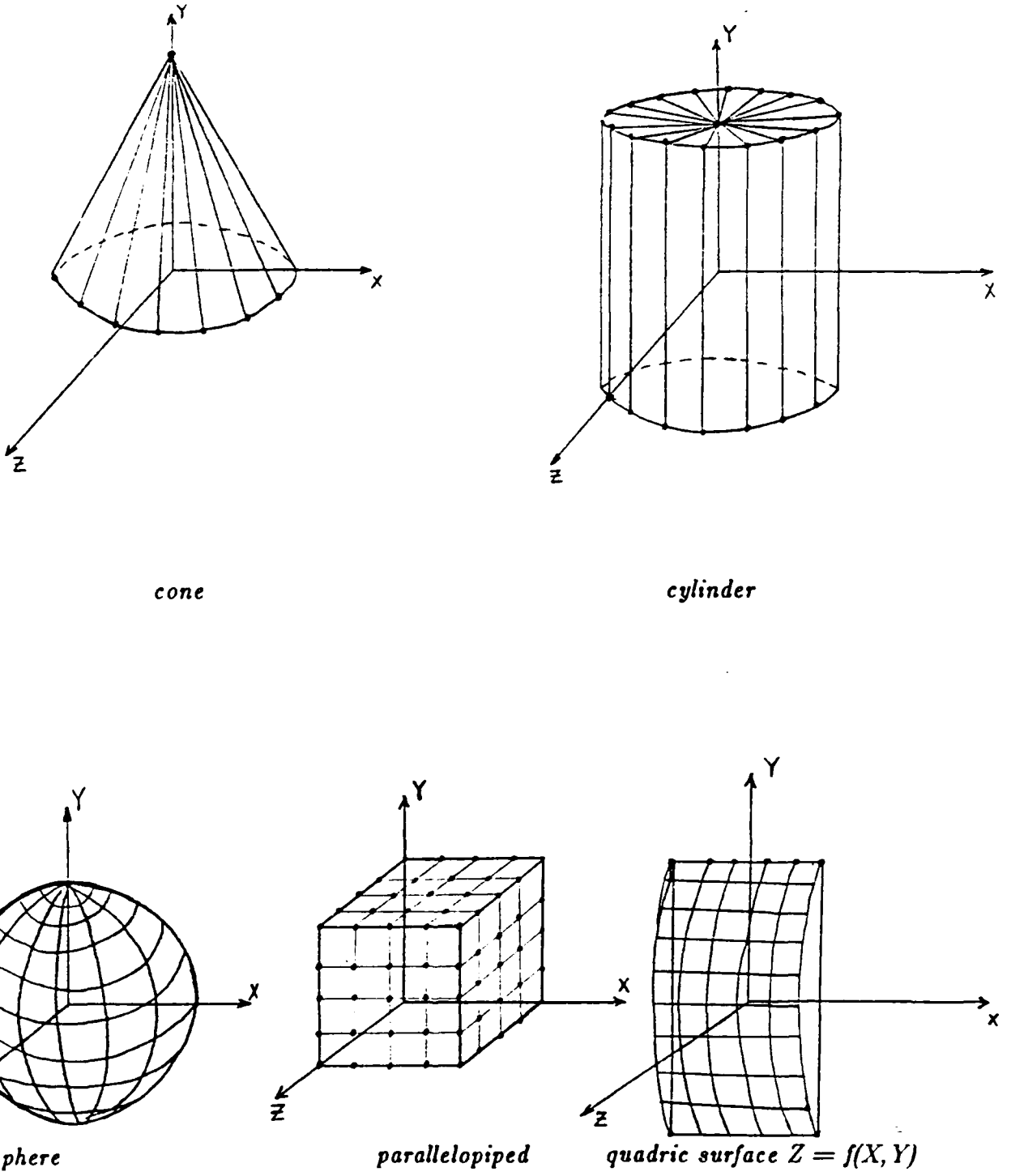

Figure 2. Control point structure for graphical primitives

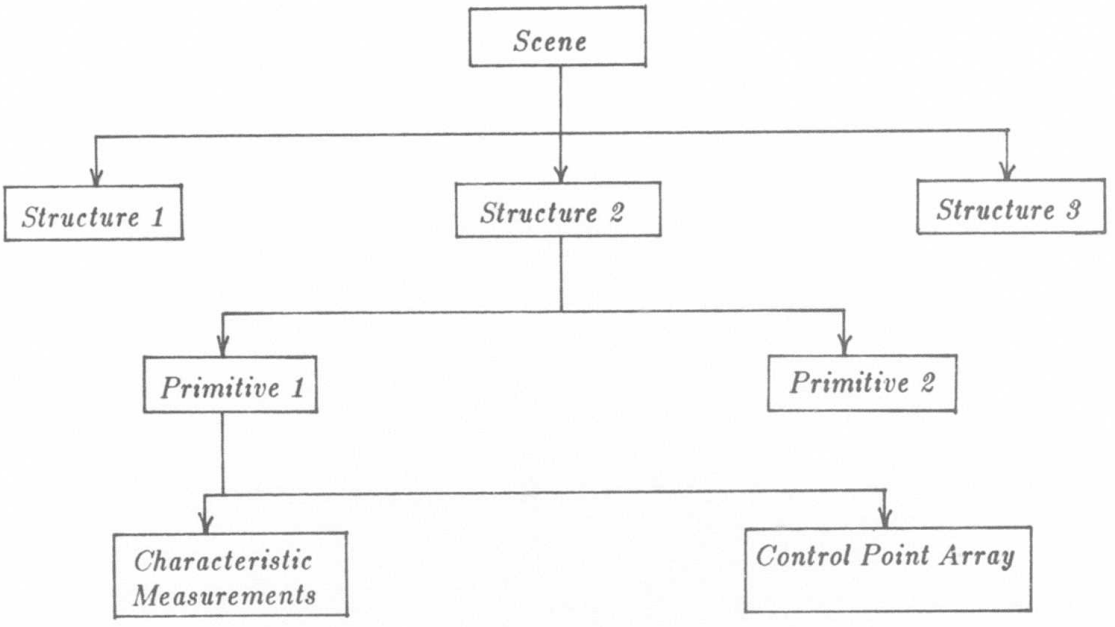

Figure 3. Hierarchical development of scene

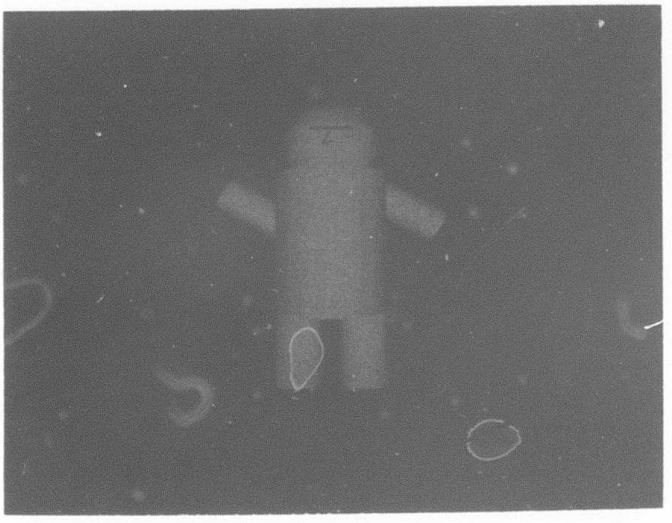

Figure 4. Example of complex object

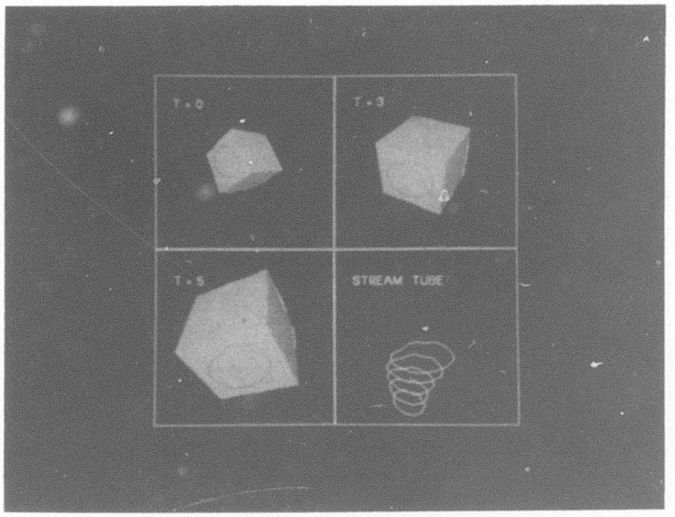

Figure 5. Contour evolution and the stream tube

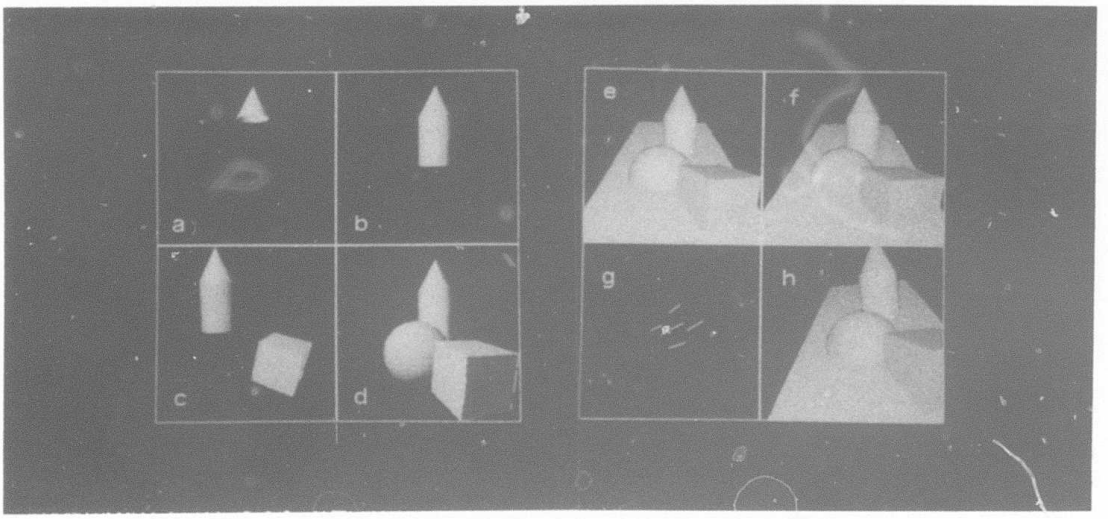

Figure 6. Typical scene being set up

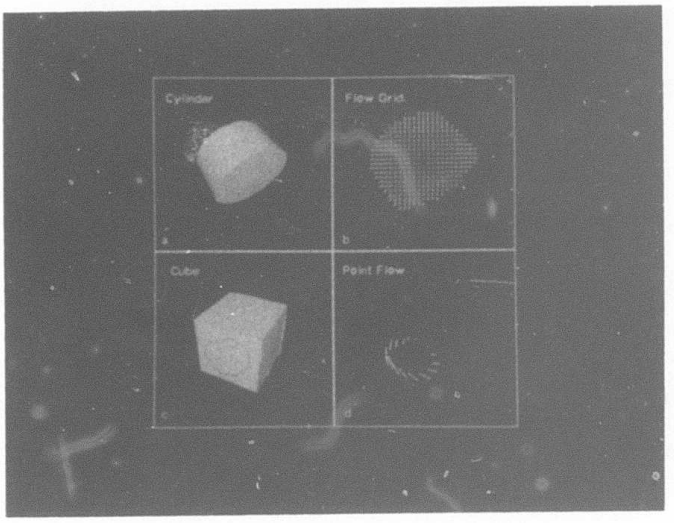

Figure 7. a) Cylinder .
b) Flow grid. Space motion is translational only.
c) Cube with contour.
d) Flow at contour vertices. Space motion is rotational and translational.

REFERENCES

[1] J.Foley and A.van Dam, **Fundamentals of Interactive Computer Graphics,** Addison-Wesley, 1982.

[2] H. Goldstein, **Classical Mechanics,** Addison-Wesley, 1959.

[3] W. Newman and R. Sproull, **Principles of Computer Graphics,** 2nd edn; McGraw-Hill, New York, 1979.

[4] H. Hubschman and S. Zucker, "Frame-to-Frame coherence and the hidden surface computation : constraints for a convex world", SIGGRAPH '81 Proceedings, published as Computer Graphics 15(3), pp. 45-54, August, 1981.

[5] K. Sugihara, "A Robust Description of Time-Varying Scenes for Computer Animation", Computers & Graphics, 7(3), pp. 277-284, 1984.

[6] I.E.Sutherland, R.F. Sproull, and R.A. Schumaker, "A characterization of ten hidden surface algorithms", Computing Surveys, 6(1), pp. 1-54, 1974.

[7] A.M. Waxman, "An Image Flow Paradigm", Proc. of Workshop on Computer Vision: Repesentation and Control, Annapolis, pp. 49-57, 1984.

[8] A.M. Waxman and S.S. Sinha, "Dynamic Stereo from Multiple Image Flows", University of Maryland, Center for Automation Research Technical Report (in preparation).

[9] A.M. Waxman and S. Ullman, "Surface Structure and 3-D Motion from Image Flow: A Kinematic Analysis", University of Maryland, Center for Automation Research Technical Report, 1983.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN IMAGE FLOW SIMULATOR		5. TYPE OF REPORT & PERIOD COVERED Technical; July 1984 6. PERFORMING ORG. REPORT NUMBER CAR-TR-71; CS-TR-1417
7. AUTHOR(s) Sarvajit S. Sinha Allen M. Waxman		8. CONTRACT OR GRANT NUMBER(s) DAAK70-83-K-0018
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Automation Research University of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Night Vision Lab Ft. Belvoir, VA 22060		12. REPORT DATE July 1984 13. NUMBER OF PAGES 40
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer vision
Image understanding
Time-varying imagery
Optical flow

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The analysis of time-varying images is currently of great interest in computer vision. There has been a deal of work recently in the study of the 2-D image flow produced due to space motion, and the recovery of the object's 3-D structure and space motion from the flow. This report details the reserve process implemented in the form of an Image Flow Simulator; from a knowledge of structure and motion, to display the 2-D image sequence and associated flow.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

This 3-D graphics animation package simulates motion of objects through space and also the evolution of surface contours through time. The graphics algorithms for projection clipping, hidden surface removal, shading and animation are described in this report.